Vers un rapprochement de l'IDM et de la compilation

Clément Guy, Steven Derrien, Benoit Combemale, Jean-Marc Jézéquel

▶ **To cite this version:**

Clément Guy, Steven Derrien, Benoit Combemale, Jean-Marc Jézéquel. Vers un rapprochement de l'IDM et de la compilation. Journées sur l'Ingénierie Dirigée par les Modèles, Jun 2011, Lille, France. inria-00601670

HAL Id: inria-00601670

https://inria.hal.science/inria-00601670

Submitted on 20 Jun 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Vers un rapprochement de l'IDM et de la compilation

Clément Guy, Steven Derrien, Benoît Combemale et Jean-Marc Jézéquel

Université de Rennes 1, IRISA/INRIA, Centre Inria Rennes - Bretagne Atlantique
Campus Universitaire de Beaulieu, 35042 Rennes Cedex, France
prénom.nom@irisa.fr

RÉSUMÉ. *L'ingénierie dirigée par les modèles (IDM) s'intéresse à la définition de systèmes complexes par leur représentation et leur abstraction à l'aide de langages de modélisation dédiés à un domaine. Le domaine de la compilation s'intéresse quant à lui à des analyses et à des traitements complexes sur des structures de données (depuis les analyses lexicale et syntaxique jusqu'à l'optimisation par rapport à la cible d'exécution). Motivées par des objectifs initiaux différents, ces deux communautés ont longtemps mené des travaux qui apparaissent aujourd'hui comme complémentaires et qui convergent vers le domaine de l'« ingénierie des langages ». Ce domaine aborde toutes les phases du cycle de vie d'un langage, depuis sa définition jusqu'à son usage. Pour cela elle regroupe des besoins en terme de développement et d'outillage des langages. Dans cet article nous analysons les apports de l'IDM et de la compilation à l'ingénierie des langages. Nous dressons ensuite un panorama des verrous à lever pour permettre une fertilisation croisée de ces deux communautés.*

ABSTRACT. *Model-driven engineering (MDE) addresses the definition of complex systems through their representation and abstraction using domain specific modeling languages (DSML). On the other hand, research in compilers takes an interest in complex analyses and transformations (from lexical and syntactical analyses to target specific optimization). Initially led by distinct goals, these two communities have had contributions which appear today as complementary each other and are converging towards the field of "software language engineering". Such a field addresses all the stages of a language lifecycle, from its definition to its use. It gathers needs in terms of language development and language tooling. In this paper we first analyze contributions of MDE and compilation to software language engineering. Then, we draw up an overview of the challenges ahead to enable a cross-fertilization between this two communities.*

MOTS-CLÉS : *IDM, Compilation, Ingénierie des langages*

KEYWORDS: *MDE, Compilation, Software Language Engineering*

1re soumission à *IDM*, le 9 mai 2011

1. Introduction

L'informatique a toujours mis en balance le traitement et la représentation de l'information (Wirth, 1978). Ce phénomène se retrouve dans la dualité entre les communautés de la compilation et de l'ingénierie dirigée par les modèles (IDM).

La communauté de la compilation s'attache à offrir des techniques avancées pour la manipulation de paradigmes de construction du logiciel (impératif, objet, etc.). Elle peut être divisée en trois grandes parties : l'analyse syntaxique et lexicale qui s'intéresse à la structure du programme et permet la création de représentations intermédiaires ; l'analyse statique et les systèmes de réécriture qui s'intéressent à la sémantique du programme ; et la compilation optimisante, qui est en lien avec le support d'exécution.

L'IDM est pour sa part le résultat de l'évolution des paradigmes de construction du logiciel pour maîtriser la complexité croissante des systèmes informatiques. Elle prône l'utilisation des modèles comme entités de premier ordre dans le développement de tels systèmes. Les modèles permettent d'abstraire la complexité du système en se concentrant sur une préoccupation particulière.

Les préoccupations de ces deux communautés se retrouvent dans le domaine émergent de l'ingénierie des langages qui inclut la définition, l'implémentation, l'outillage et l'utilisation des langages (Favre *et al.,* 2009). Cette ingénierie peut tirer partie de l'IDM et de la compilation de manière à maîtriser la complexité de la description et de l'outillage des langages. Nous présentons dans cet article les apports de ces deux communautés à l'ingénierie des langages, ainsi que les bénéfices qu'apporterait leur rapprochement.

2. Motivations pour une ingénierie des langages

Le terme de langage logiciel regroupe tous les types de langages liés au cycle de vie du logiciel et à l'ensemble de ses préoccupations (e.g., processus, IHM, sécurité, etc.) (Favre *et al.,* 2009). Plusieurs constats se dégagent autour des langages logiciels :

– Les logiciels et les préoccupations qu'ils adressent se multipliant, le nombre de langages logiciels dédiés à ces préoccupations, ne cesse d'augmenter (Kleppe, 2008). Cette augmentation amène plusieurs besoins dans la manière de traiter les langages, parmi lesquels :

1) La mise en place d'une ingénierie efficace pour la définition de langages.

2) La possibilité de raisonner au travers des différents langages utilisés pour un système logiciel pour pouvoir déterminer des propriétés sur le système global.

3) La mutualisation des traitements (e.g., analyses, vérifications, etc.) pour un ensemble de langages de manière à pouvoir les réutiliser sans avoir à les réimplémenter à partir de zéro.

– Le nombre et la complexité des traitements ne cessant également d'augmenter, il faut pouvoir assurer une cohérence entre les traitements qui sont appliqués à un

artefact de description (e.g., programme, modèle, etc.) issu d'un langage logiciel.

– Pour maîtriser la taille et la complexité des systèmes logiciels, il est préconisé de séparer les préoccupations, ce qui permet de définir des langages dédiés d'un plus haut niveau d'abstraction. Cette abstraction croissante peut rendre difficile la production d'un code machine efficace. Ceci couplé à la taille des *artefacts de description* risque de poser des problèmes de performance à l'ingénierie des langages et nécessite une attention particulière.

La séparation des préoccupations et la montée en abstraction des langages logiciels permettent de maîtriser la complexité des systèmes logiciels. Néanmoins cette complexité se retrouve dans la définition et l'outillage des langages. Nous étudions dans la suite de cet article les apports possibles des communautés de l'IDM et de la compilation, qui ont développé des expertises dans ces domaines.

3. Apports de l'IDM et de la compilation à l'ingénierie des langages

L'IDM est le résultat de l'évolution des bonnes pratiques de conception de systèmes logiciels. Elle peut ainsi apporter à l'ingénierie des langages certaines de ces bonnes pratiques :

– La définition, en termes abstraits, d'un système logiciel par des modèles, chacun correspondant à une préoccupation particulière pour le système. Cette approche se découpe en deux activités :

1) La séparation des différentes préoccupations concernant le système sous forme de modèles. Les informations nécessaires au traitement de la préoccupation choisie sont réunies dans un langage de modélisation.

2) La composition des points de vue modélisés de manière automatique ou semi-automatique pour obtenir un modèle complet du système logiciel.

– L'outillage des langages de modélisation par des approches génératives dédiées qui permettent de générer à partir des modèles un certain nombre d'artefacts (code, tests, etc.).

– La conception par contrats par l'intégration de contraintes (e.g., contraintes OCL [1]) aux langages de modélisation, qui permet de spécifier les entrées et sorties d'un traitement et peut donc assurer la cohérence entre les traitements.

– La réutilisation dans l'IDM a été abordée par la programmation générique (Cuccuru *et al.,* 2007, Moha *et al.,* 2009, de Lara *et al.,* 2010). La réutilisation de traitements est alors possible pour des langages dont la structure partage une relation d'alignement avec celle sur laquelle les traitements ont été définis.

– La définition des langages de modélisation à partir de méta-modèles, conformes au MOF [2], un métalangage défini par l'OMG. L'uniformité apportée par le MOF per-

1. Object Constraint Language : http ://www-st.inf.tu-dresden.de/ocl/
2. Meta-Object Facility : http ://www.omg.org/mof/

met de définir des outils génériques à tous les langages de modélisation comme des éditeurs, des visualiseurs ou des outils de sérialisation de modèles.

La communauté de la compilation a développé une expertise dans l'utilisation d'un certain nombre de techniques, parmi lesquelles :

– des techniques de vérification et de validation (interprétation abstraite, *model-checking*) dans les différentes phases de compilation (e.g., les analyses de flot de contrôle) ;

– des systèmes de réécriture d'arbres (e.g., TOM (Moreau *et al.,* 2003)) permettant entres autres la manipulation efficace d'arbres de syntaxe abstraite.

Ces techniques ont évidemment un usage pour un grand nombre des langages logiciels, et l'ingénierie des langages gagnerait à réutiliser l'expertise de la communauté de la compilation dans ces domaines.

La communauté de la compilation optimisante peut aussi apporter à l'ingénierie des langages un savoir-faire dans l'algorithmique complexe visant à résoudre des problèmes d'optimisation en rapport avec les plateformes d'exécution (e.g. allocation de registres, parallélisation automatique, etc. (Muchnick, 1997)) et dans les heuristiques permettant le passage à l'échelle de cette algorithmique. Cette expérience peut contribuer à l'amélioration des performances des outils de l'ingénierie des langages et du code généré par ces outils.

4. Verrous à lever pour une fertilisation croisée de l'IDM et de la compilation

Certains verrous restent cependant à lever pour tirer parti de l'expérience de l'IDM et de la compilation au sein de l'ingénierie des langages. Nous exposons dans la suite de cette section trois d'entre eux qui nous semblent particulièrement importants :

– En complément d'approches génératives permettant d'outiller automatiquement un nouveau langage (e.g., éditeur de modèles avec GMF) il est également nécessaire d'offrir des mécanismes de mutualisation des traitements sur les langages.

– Afin de travailler au niveau d'un système logiciel complet, composés d'*artefacts de description* issus de différents langages, il faut pouvoir définir des traitements inter-langages et donc raisonner sur un ensemble de langages et leurs relations.

– Les systèmes de réécriture ainsi qu'une partie des analyses utilisées en compilation travaillent sur des arbres, là où les outils apportés par l'IDM représentent les langages et les *artefacts de description* qui en sont issus comme des graphes. Il est donc nécessaire de définir un cadre permettant une adaptation de ces analyses sur les graphes.

5. Vers une gestion des langages par typage et composition

La mutualisation de traitements et les traitements inter-langages peuvent être abordés en fournissant des outils au niveau du métalangage.

Nous étudions la définition d'un système de type autorisant la substituabilité des modèles et la réutilisation des traitements définis sur les langages de modélisation. De la même manière qu'un type d'objet t peut hériter d'un autre type t', on voudrait qu'un type de modèle t_M puisse hériter d'un type de modèle t'_M. Ainsi, définir un nouveau langage pourrait se faire en étendant par héritage des langages pré-existants. De cette manière le nouveau langage hériterait des transformations définies sur ses "super-langages".

Des travaux existants abordent de manière complémentaire le typage dans l'IDM. (Vignaga *et al.,* 2009) s'intéressent au typage des transformations de modèles sans ouvrir la possibilité de substituabilité des types de modèles. Un système de type pour les modèles a également été proposé par (Steel *et al.,* 2007) pour autoriser une telle substituabilité à l'aide de généricité. Par ailleurs, des possibilités d'utiliser la programmation générique dans l'IDM ont également été développées dans (Cuccuru *et al.,* 2007) et (de Lara *et al.,* 2010).

De manière orthogonale à la capitalisation de traitements, il est nécessaire de pouvoir définir des traitements inter-langages (e.g., consistence et interopérabilité) et donc des relations entre langages. Nous voulons aborder ce problème par la composition de langages et son intégration au niveau du métalangage.

Les macromodèles (Salay *et al.,* 2008) et les mégamodèles (Favre *et al.,* 2004) définissent des relations entre modèles et les relations entre leurs éléments. Les premiers sont utilisés dans un but de conception, tandis que l'implémentation des deuxièmes au sein d'AM3 [3] est utilisée pour la navigation et la traçabilité des transformations de modèles. Il existe également une approche formelle basée sur les TGG (*Triple Graph Grammars*) pour spécifier les transformations de modèle à modèle et étendue pour le *model matching* et la traçabilité (Guerra *et al.,* 2010).

Nous voulons intégrer le typage et la composition de langages au sein d'un métalangage, afin de prendre en compte les traitements aussi bien intra qu'inter-langages et ainsi autoriser leur réutilisation. En effet, si des méthodes existent déjà pour la réutilisation de traitements intra-langages et pour la définition de traitements inter-langages, elles ne sont pour le moment pas unifiée au sein d'un même système. Pour cela, nous proposons d'étendre le typage de Steel *et al.* en intégrant au métalangage MOF la possibilité de définir des relations (héritage, consistence, etc.) entre langages.

6. Conclusion

L'ingénierie des langages est un domaine émergent qui s'intéresse à l'étude systématique des langages logiciels. Nous étudions dans cet article les apports respectifs des communautés de l'IDM et de la compilation à cette nouvelle ingénierie.

Bien qu'issues de motivations différentes, les communautés de l'IDM et de la compilation ont développé une expertise complémentaire. L'IDM prône la séparation des

3. AtlanMod MegaModel Management : http ://www.eclipse.org/gmt/am3/

préoccupations et la capitalisation de l'expérience au sein de langages dédiés à une préoccupation. Pour cela, elle offre un cadre conceptuel et technologique permettant de définir de nouveaux langages et de les outiller. Par ailleurs, la communauté de la compilation a développé une expertise dans la définition de chaînes de compilation complexes, intégrant différentes analyses et optimisations.

La fertilisation croisée des communautés de l'IDM et de la compilation peut contribuer à l'ingénierie des langages en offrant un cadre conceptuel, des outils et une expérience dans la définition de nouveaux langages et de leurs outils associés. Elle pose néanmoins de nouveaux verrous. En particulier, la multiplication des langages manipulés dans le cycle de vie d'un système informatique nous impose de pouvoir raisonner sur des ensembles de langages.

Nous proposons d'unifier les notions de liens entre langage au niveau du métalangage, qu'ils s'agissent de relations d'héritage permettant la réutilisation ou de relations de composition permettant de définir des traitements inter-langages.

7. Bibliographie

Cuccuru A., Mraidha C., Terrier F., Gérard S., « Templatable metamodels for semantic variation points », ECMDA-FA, Springer-Verlag, p. 68-82, 2007.

de Lara J., Guerra E., « Generic meta-modelling with concepts, templates and mixin layers », MODELS, Springer-Verlag, p. 16-30, 2010.

Favre J.-M., Gasevic D., Lämmel R., Winter A., « Guest Editors' Introduction to the Special Section on Software Language Engineering », *IEEE TSE*, vol. 35, n° 6, p. 737-741, 2009.

Favre J.-M., NGuyen T., « Towards a Megamodel to Model Software Evolution through Transformations », *SETRA Workshop, Elsevier ENCTS*, p. 59-74, 2004.

Guerra E., de Lara J., Kolovos D. S., Paige R. F., « Inter-modelling : from theory to practice », MODELS, Springer-Verlag, p. 376-391, 2010.

Kleppe A., *Software Language Engineering : Creating Domain-Specific Languages Using Metamodels*, 1 edn, Addison-Wesley Professional, 2008.

Moha N., Mahé V., Barais O., Jézéquel J.-M., « Generic Model Refactorings », MODELS, Springer-Verlag, p. 628-643, 2009.

Moreau P.-E., Ringeissen C., Vittek M., « A pattern matching compiler for multiple target languages », CC, Springer-Verlag, p. 61-76, 2003.

Muchnick S. S., *Advanced compiler design and implementation*, Morgan Kaufmann Publishers Inc., 1997.

Salay R., Mylopoulos J., Easterbrook S., « Managing Models through Macromodeling », ASE, IEEE Computer Society, p. 447-450, 2008.

Steel J., Jézéquel J.-M., « On Model Typing », *SoSyM*, vol. 6, n° 4, p. 401-414, 2007.

Vignaga A., Jouault F., Bastarrica M. C., Brunelière H., « Typing in Model Management », ICMT, Springer-Verlag, p. 197-212, 2009.

Wirth N., *Algorithms + Data Structures = Programs*, Prentice Hall PTR, 1978.